Original Article

e-ISSN: 2774-2016 - https://journal.itera.ac.id/index.php/indojam/
p-ISSN: 2774-2067

Received 13th January 2025
Accepted 9th April 2025
Published 25th April 2025

Open Access

DOI:
https://doi.org/10.35472/indojam.v5i1.2104

Quantitative Performance Analysis of Spring-Mass-Damper Control Systems: A Comparative Implementation in Python and R

Sandy H. S. Herho [*a], Siti N. Kaban [b]

[a] *Department of Earth and Planetary Sciences, University of California, Riverside, CA, USA*

[b] *Offshore Engineering Research Group, Bandung Institute of Technology (ITB), Bandung, Indonesia*

** Corresponding E-mail: sandy.herho@email.ucr.edu*

Abstract: The numerical simulation and control of spring-mass-damper (SMD) systems offer critical insights into dynamical systems and computational methodologies. This study provides a comprehensive comparative analysis of implementing SMD systems across two prominent open-source scientific computing platforms: Python and R. By examining both open-loop and closed-loop system configurations, the research investigates the computational performance, numerical accuracy, and implementation characteristics of these platforms. Utilizing an idealized one-dimensional SMD system with a Proportional-Integral-Derivative (PID) controller, the study conducted extensive numerical simulations and statistical performance analyses. Results revealed Python's significant advantages in execution speed, achieving up to 63.57% reduction in runtime for controlled system simulations, while R demonstrated superior consistency in execution and memory usage. The controlled system demonstrated exceptional performance, with a final position error of merely 0.4% and enhanced damping characteristics. This work not only bridges theoretical stability analysis with empirical performance insights but also promotes reproducibility and transparency in computational dynamics research by leveraging open-source platforms.

Keywords: *control systems, cross-platform implementation, numerical methods, performance analysis, spring-mass-damper dynamics*

Abstrak: Simulasi numerik dan pengendalian sistem pegas-massa-peredam (Spring-Mass-Damper/SMD) merupakan fondasi penting dalam bidang keilmuan sistem dinamis dan metodologi komputasi. Penelitian ini menyajikan analisis komparatif yang komprehensif mengenai implementasi sistem SMD pada dua platform komputasi ilmiah sumber terbuka yang terkemuka, yakni Python dan R. Melalui pengkajian konfigurasi sistem loop terbuka dan loop tertutup, penelitian ini menginvestigasi kinerja komputasi, akurasi numerik, dan karakteristik implementasi dari kedua platform tersebut. Dengan memanfaatkan sistem SMD satu dimensi yang diidealkan dengan pengendali Proporsional-Integral-Derivatif (PID), studi ini melakukan simulasi numerik ekstensif dan analisis kinerja statistik. Hasil penelitian mengungkapkan keunggulan signifikan Python dalam kecepatan eksekusi, mencapai pengurangan waktu eksekusi hingga 63,57% untuk simulasi sistem terkendali, sementara R menunjukkan konsistensi yang lebih baik dalam eksekusi dan penggunaan memori. Sistem terkendali mendemonstrasikan kinerja yang luar biasa, dengan kesalahan posisi akhir hanya sebesar 0,4% dan karakteristik peredaman yang ditingkatkan. Penelitian ini tidak hanya menjembatani analisis stabilitas teoretis dengan pandangan empiris, tetapi juga mendorong reproduktibilitas dan transparansi dalam penelitian dinamika komputasi dengan memanfaatkan platform terbuka.

Kata Kunci: *sistem kendali, implementasi lintas platform, metode numerik, analisis kinerja, dinamika pegas-massa-peredam*

Introduction

The numerical analysis and computational implementation of spring-mass-damper (SMD) systems present rich opportunities for investigating fundamental questions in dynamical systems and numerical methods. These systems, while

Original Article

conceptually straightforward, embody essential characteristics of more complex dynamical systems and serve as valuable benchmarks for evaluating numerical methods and computational frameworks. With the rise of open science practices and increasing demands for research transparency [21], the choice of computational platforms becomes particularly significant in ensuring both mathematical rigor and reproducibility.

The mathematical structure of SMD systems manifests across multiple scales of computational physics and engineering analysis, from simple mechanical oscillators to sophisticated control systems. Our investigation encompasses both open-loop and closed-loop configurations, allowing us to examine how different numerical methods handle varying degrees of system complexity. In the current landscape, where open science platforms are reshaping scientific communication [22], implementing these systems in accessible, and transparent environments becomes crucial for validating both theoretical predictions and numerical approximations.

Among open-source platforms, Python and R have gained significant adoption in scientific computing [42, 48], each offering distinct approaches to numerical computation and algorithm implementation. Contemporary scientific computing increasingly relies on these open-source platforms, which align with broader movements toward research transparency and reproducibility.

Python's scientific computing ecosystem, built around NumPy and SciPy, exemplifies how open-source tools can enable sophisticated numerical computations while maintaining accessibility. The platform's implementation of variable-step, variable-order methods provide particular insight into handling stiff differential equations and adaptive error control.

Similarly, R's statistical computing framework, particularly through packages like deSolve, demonstrates how community-driven development can produce robust tools for complex mathematical modeling, offering unique perspectives on numerical stability and error propagation. The computational challenges become particularly evident when implementing variable-step, variable-order methods for these systems, especially in the context of PID control where system stiffness can vary significantly.

As scientific practices evolve toward greater openness and accessibility [23], the ability to examine and validate implementation details becomes crucial. Both Python and R provide transparent implementations of numerical methods, allowing researchers to understand and verify the underlying algorithms - a key advantage for studying numerical stability, convergence properties, and error accumulation in long-time integration.

The significance of this comparative study extends beyond mere platform evaluation, addressing fundamental questions in computational mathematics and numerical analysis. In an era where research integrity increasingly depends on computational reproducibility [22], understanding the relative strengths and limitations of different open-source implementations becomes crucial.

First, the choice between Python and R affects not only computational performance but also numerical accuracy and stability, particularly in handling stiff systems and adaptive step size control. Second, as open-source platforms continue to evolve, systematic comparisons help inform both theoretical understanding of numerical methods and their practical implementation.

Third, with the growing emphasis on research transparency, these platforms provide an ideal foundation for studying how different numerical schemes behave in both open and closed-loop dynamical systems, contributing to both applied mathematics and computational science.

Method

2.1 Mathematical Formulation

Original Article

Our analysis began with an idealized one-dimensional SMD system, consisting of a point mass $m = 100\,\text{kg}$ coupled to both an ideal linear spring of stiffness $k = 50\,\text{N/m}$ and a viscous damper with coefficient $d = 50\,\text{Ns/m}$, as illustrated in Figure 1. The system's idealization encompassed several key simplifications: the mass behaved as a point particle devoid of rotational dynamics, the spring exhibited perfectly linear behavior following Hooke's law with negligible mass, the damper provided purely viscous damping, and the system operated in the absence of friction or additional constraints. All connections between components were considered rigid and massless, ensuring that the system's behavior was governed solely by the interplay of inertial, elastic, and dissipative forces.

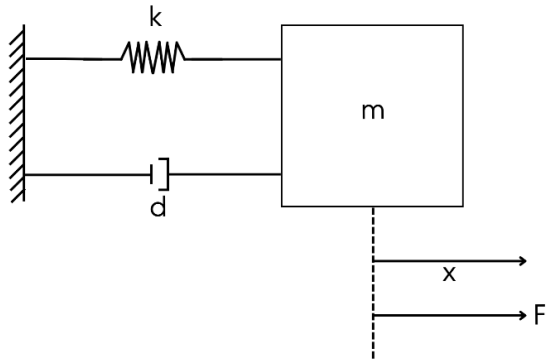

Figure 1. Physical representation of the idealized SMD system showing coordinate system and force components. The origin was set at the spring's equilibrium position, with positive displacement defined in the rightward direction.

The mathematical model emerged directly from Newton's Second Law, which stated that the sum of forces acting on the mass equalled the product of mass and acceleration. In our system, three distinct forces acted on the mass: the spring force $\left(\text{F}_\text{s} = -\text{kx(t)}\right)$, the damping force proportional to velocity $\left(\text{F}_\text{d} = -\text{d}\dot{\text{x}}\text{(t)}\right)$, and an external force F(t). The negative signs in the spring and damping forces indicated their opposition to displacement and motion, respectively. Applying Newton's Second Law and substituting these forces yielded the fundamental equation of motion:

$$\text{m}\ddot{\text{x}}\text{(t)} = \text{F(t)} - \text{kx(t)} - \text{d}\dot{\text{x}}\text{(t)}\,, \tag{1}$$

which, upon substituting our system parameters, became:

$$100\ddot{\text{x}}\text{(t)} = 50\dot{\text{x}}\text{(t)} + 50\text{x(t)} + \text{F(t)}\,. \tag{2}$$

To facilitate modern control analysis and implementation, we transformed this second-order differential equation into state-space form. The transformation introduced a state vector $\text{x(t)} = [x_1(t), x_2(t)]^T$, where x_1 represented position and x_2 represented velocity. This choice of states provided clear physical insight: x_1 captured the system's configuration while x_2 described its instantaneous motion. The resulting state equations took the form:

$$\begin{aligned} \dot{\text{x}}_1 &= x_2, \\ \dot{\text{x}}_2 &= -0.5x_1 - 0.5x_2 + 0.01F(t)\,, \end{aligned} \tag{3}$$

which could be expressed in the canonical matrix form:

$$\begin{bmatrix} \dot{\text{x}}_1 \\ \dot{\text{x}}_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 \\ -0.5 & -0.5 \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ 0.01 \end{bmatrix}}_{B} F(t)\,. \tag{4}$$

For precise position control, we implemented a Proportional-Integral-Derivative (PID) controller with carefully tuned gains: $K_p = 200\,\text{N/m}$ for proportional action, $K_i = 50\,\text{N/ms}$ for integral action, and $K_d = 100\,\text{N.s/m}$ for derivative action. These PID controller gains represent the normalized control parameters, which were scaled relative to the system mass of 100 kg for direct implementation in the state-space formulation. This normalization simplifies the mathematical model while preserving the physical interpretation of the controller behavior. The PID controller gains were systematically tuned following the frequency-domain design methodology outlined by [20],

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

which optimizes performance based on phase and gain margins. Initial values were derived using the Ziegler-Nichols method, followed by fine-tuning to achieve the desired response characteristics in terms of settling time and overshoot for the specific SMD configuration. Given a desired position trajectory $x_d(t)$, the control law synthesized the input force as:

$$F(t) = K_p[x_d(t) - x_1(t)] + K_i \int_0^t [x_d(\tau) - x_1(\tau)]d\tau + K_d[\dot{x}_d(\tau) - \dot{x}_2(\tau)]. \quad \textbf{(5)}$$

The integral action in the controller necessitated augmenting our state space with an additional state variable $x_3 = \int_0^t [x_d(\tau) - x_1(\tau)]d\tau$ representing the accumulated position error. This augmentation led to the complete closed-loop dynamics:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -2.5 & -1.5 & -0.5 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.01 \\ 0 \end{bmatrix} x_d(\tau) \, . \quad \textbf{(6)}$$

The closed-loop system in equation (6) incorporates both the original system dynamics and the PID control law, where the state x_3 represents the accumulated position error. The resulting state matrix reflects the augmented system dynamics including controller effects. The complete system architecture, including both open-loop and closed-loop configurations, was illustrated in Figure 2. This representation explicitly showed the signal flow and system interconnections, highlighting the fundamental difference between direct force input and feedback control strategies.

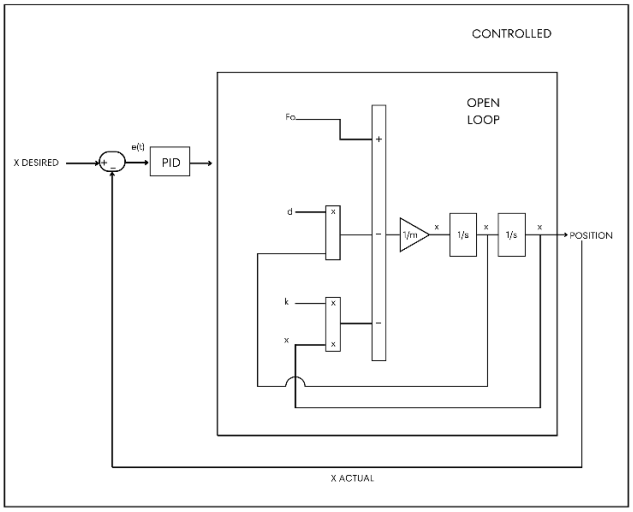

Figure 2. Block diagram representation of the SMD system: Open-loop configuration showing direct force input with system transfer function and closed-loop configuration illustrating PID control architecture with complete signal flow paths and transfer functions.

2.2 Numerical Implementation

The numerical solution of our SMD system required careful consideration of both accuracy and stability, particularly given the potential stiffness introduced by the control parameters. Our implementation leveraged the Python scientific computing ecosystem, specifically employing NumPy [24] for array operations and the Livermore Solver for Ordinary Differential Equations with Automatic method switching (LSODA) through SciPy's integrate module [25]. For data manipulation and storage of results, we utilized the pandas library [43], which provided efficient DataFrame structures for handling time series data and simulation outputs.

The core of our numerical approach lay in the transformation of our continuous-time system into a form suitable for computational solution. For the open-loop system, we began with the state-space representation derived in our mathematical formulation. The solver required a system model function that returned the state derivatives:

$$\frac{d}{d_t}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = f(x, t, F) \, , \quad \textbf{(7)}$$

where $x = [x_1, x_2]^T$ represented our state vector, and F was the input force. This formulation expanded to our specific parameter values:

$$f(x,t,F) = \begin{bmatrix} x_2 \\ \frac{F - 50x_1 - 50x_2}{100} \end{bmatrix}. \quad \textbf{(8)}$$

The LSODA algorithm employed an adaptive strategy, automatically switching between methods optimal for stiff and non-stiff regions of the solution. In numerical analysis, stiff regions of differential equations are characterized by solution components that change at vastly different rates, causing standard explicit methods to require prohibitively small step sizes for stability. In our SMD system, stiffness emerges particularly during rapid transitions in the control force and in the controlled system where the eigenvalue spread increases due to controller action. Conversely, non-stiff regions exhibit more uniform rates of change across solution components, allowing larger step sizes without stability concerns. This distinction becomes especially important when simulating mechanical systems under high-gain feedback control. For non-stiff regions, it utilized Adams-Moulton methods of orders 1-12, which took the general form:

$$x_{n+1} = x_n + h\sum_{i=0}^{k} \beta_i \mathrm{f}_{n-i}\,, \quad \textbf{(9)}$$

where h represented the step size and β_i were the Adams-Moulton method coefficients, which are determined by the specific order of the method being used. These coefficients are derived from the integration of the Lagrange interpolation polynomials of the derivative function. For stiff regions, the solver switched to backward differentiation formulas (BDF):

$$\sum_{i=0}^{k} \alpha_i \mathrm{x}_{n+1-i} = h\beta_0 \mathrm{f}_{n+1}\,, \quad \textbf{(10)}$$

where α_n and β_0 were the BDF method coefficients, with α_I controlling the weighting of previous solution values and β_0 determining the contribution of the current derivative estimate. These coefficients are selected to optimize stability and accuracy properties for stiff systems.

The simulation time domain was discretized with careful attention to numerical stability and accuracy requirements. We employed a base step size of $h = 0.01$ seconds over a simulation interval [0, 100] seconds for the open-loop system and [0, 20] seconds for the controlled system. This discretization resulted in:

$$t_i = t_0 + ih, \;\; i = 0, 1, \ldots, N\,, \quad \textbf{(11)}$$

where $N = (t_f - t_0)/h$ represented the total number of time steps.

The input force profile was generated through a carefully designed transition function to avoid numerical artifacts. For a step input from F_1 to F_2 curring between times t_1 and t_2, we implemented a linear transition:

$$\begin{cases} F_1, & t < t_1 \\ F_1 + \dfrac{F_2 - F_1}{t_2 - t_1}(t - t_1), & t_1 \le t \le t_2, \\ F_2, & t > t_2, \end{cases} \quad \textbf{(12)}$$

where F_1 represents the initial force magnitude (0 N) before the transition begins at time t_1, and F_2 represents the final force magnitude (50 N) after the transition completes at time t_2. This linear transition in force magnitude between times $t_1 = 4$ s and $t_2 = 5$ s provides a smooth input that avoids exciting high-frequency dynamics that could occur with an instantaneous step input.

For the closed-loop system, the numerical implementation became more intricate due to the PID control law. The augmented state vector $[x_1, x_2, x_3]^T$ required modification of our system model to incorporate the integral term:

$$f(x,t,x_d) = \begin{bmatrix} x_2 \\ \frac{200(x_d - x_1) + 50x_3 - 100x_2 - 50x_1}{100} \\ x_d - x_1 \end{bmatrix}. \quad \textbf{(13)}$$

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

The second component of this system incorporates the complete PID control law from equation (5), properly accounting for all gain terms and state variables. The term $[200(x_d - x_1) + 50x_3 - 100x_2 - 50x_1]/100$ represents the acceleration resulting from the combined effect of the PID controller force and the system's internal forces, normalized by the mass. The numerical solver handled this augmented system with the same adaptive strategy, but special care was taken in the initialization and update of the integral term to prevent numerical drift. The desired position trajectory $x_d(t)$ followed a similar smooth transition profile, but with values scaled to position units: $x_{d,\text{initial}} = 0$ m to $x_{d,\text{final}} = 1$ m m between $t_1 = 2\ s$ and $t_2 = 2.5$ s.

Error control in our numerical solution was maintained through careful monitoring of local truncation error at each step:

$$LTE_n = \|x_{n+1} - \tilde{x}_{n+1}\| \leq tol\ , \tag{14}$$

where $\tilde{x}_{n+1}$ represented a higher-order approximation used for error estimation. In our notation, subscripts refer to time discretization, not state components. Specifically, x_n denotes the complete state vector at time step n, while x_{n+1} represents the state vector at the subsequent time step $n+1$. The components of the state vector at any time step are indicated by superscripts or explicit notation (e.g., x_1, x_2 for position and velocity states). The step size was adaptively adjusted based on this error estimate:

$$h_{new} = h_{old}\left(\frac{tol}{LTE_n}\right)^{\frac{1}{p}}, \tag{15}$$

with p denoting the current method order and tol=10^{-8} as our specified tolerance.

The complete implementation was organized into modular Python classes, following object-oriented programming principles, which facilitated easy modification and extension of the system model and control algorithms. All numerical computations were performed using Python, with NumPy, SciPy, and pandas. For comparative validation and robustness testing, parallel implementations were developed in R using the deSolve package [26] for differential equation solving and the tidyverse ecosystem [27] for data manipulation and visualization. The results from both implementations showed excellent agreement, providing confidence in the numerical accuracy of our solutions.

2.3 Stability Analysis

Our stability analysis methodology incorporated multiple theoretical frameworks to thoroughly assess both open-loop and closed-loop system behavior. Following [1], we employed three complementary approaches of eigenvalue analysis, Routh-Hurwitz criterion, and Bounded-Input, Bounded-Output (BIBO) stability assessment, with each method providing unique insights into the system's dynamic characteristics.

The first stage of our analysis centered on eigenvalue decomposition. For a linear time-invariant system, stability required that all eigenvalues had negative real parts [2]. The analysis began with our open-loop system matrix:

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{d}{m} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -0.5 & -0.5 \end{bmatrix}, \tag{16}$$

from which we formulated the characteristic equation through determinant evaluation:

$$\det(s\mathbf{I} - \mathbf{A}) = \begin{vmatrix} s & -1 \\ 0.5 & s + 0.5 \end{vmatrix} = s^2 + \frac{d}{m}s + \frac{k}{m} = s^2 + 0.5s + 0.5 = 0\ , \tag{17}$$

Following [3], we determined two fundamental parameters characterizing the system's behavior. The natural frequency ω_n represented the system's undamped oscillation rate, while the damping ratio ζ characterized the decay rate of oscillations:

$$\omega_n = \sqrt{\frac{k}{m}} = \sqrt{0.5}\ , \quad \zeta = \frac{d}{2\sqrt{mk}} = \frac{50}{2\sqrt{100.50}} = 0.354\ . \tag{18}$$

Original Article

The nature of system damping played a crucial role in determining the transient response characteristics. As derived by [1], the damping ratio $\zeta = 0.354$ indicated an underdamped system, as it fell within the range $0 < \zeta < 1$. In such cases, the system's free response exhibited oscillatory behavior with exponential decay. The complete solution took the form:

$$x(t) = Ae^{-\zeta\omega_n t}\cos(\omega_d t + \phi) \tag{19}$$

where $\omega_d = \omega_n\sqrt{1-\zeta^2}$ represented the damped natural frequency, and A and ϕ were determined by initial conditions. This underdamped characteristic arose from our physical parameters, with the mass-spring-damper combination producing complex conjugate eigenvalues $\lambda_{1,2} = -\zeta\omega_n \pm i\omega_d = -0.25 \pm 0.661i$.

Asymptotic stability analysis followed the theoretical framework of [2], which required examination of the system's long-term behavior as $t \to \infty$. For our linear system, asymptotic stability demanded that all eigenvalues had strictly negative real parts. The real component of our eigenvalues, $-\zeta\omega_n = -0.25$, being negative, ensured exponential decay of any initial conditions or disturbances. This stability manifested through the exponential term $e^{-\zeta\omega_n t}$ in the system response, which approached zero as time increased.

The Routh-Hurwitz stability criterion [4] provided an algebraic method for determining stability without explicit eigenvalue calculation. For our second-order characteristic equation:

$$a_2 s^2 + a_1 s + a_0 = s^2 + 0.5s + 0.5 = 0\,, \tag{20}$$

we constructed the Routh array and examined its first column entries:

$$\begin{array}{c|cc} s^2 & 1 & 0.5 \\ s^1 & 0.5 & 0 \\ s^0 & 0.5 & \end{array}\,. \tag{21}$$

The stability conditions derived from the Routh array required all elements in its first column to be positive. Following [5], this translated to the requirements $a_2 > 0$, $a_1 > 0$, and $a_0 > 0$, which were all satisfied in our system, confirming stability from an algebraic perspective.

The BIBO stability assessment followed [6] methodology, focusing on the system's transfer function:

$$G(s) = \frac{1}{ms^2+ds+k} = \frac{0.01}{s^2+0.5s+0.5}\,. \tag{22}$$

This approach required examination of both the positivity of denominator coefficients and the Hurwitz stability of the denominator polynomial. Our transfer function satisfied both conditions, as all denominator coefficients were positive and matched the previously analyzed characteristic equation.

For the closed-loop analysis with PID control, we extended these methodologies to the augmented third-order system. The characteristic equation expanded to include controller parameters:

$$s^3 + \frac{d+K_d}{m}s^2 + \frac{k+K_p}{m}s + \frac{K_i}{m} = 0\,, \tag{23}$$

necessitating an expanded Routh array for stability analysis:

$$\begin{array}{cc} s^3 & 1 \qquad\qquad \dfrac{k+K_p}{m} \\ s^2 & \dfrac{d+K_d}{m} \qquad\quad \dfrac{K_i}{m} \\ s^1 & \dfrac{(d+K_d)(k+K_p)-mK_i}{m(d+K_d)} \qquad 0 \\ s^0 & \dfrac{K_i}{m} \end{array}\,. \tag{24}$$

The relationship between damping and asymptotic stability was further illuminated through the root locus analysis detailed by [5]. The characteristic equation roots, or system poles, lying in the left half of the complex plane ($\mathrm{Re}(\lambda) < 0$) guaranteed that any oscillations would eventually decay. The distance of these poles from the imaginary axis, determined by $\zeta\omega_n$, dictated the

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

decay rate, while their imaginary components $\pm\omega_d$ determined the oscillation frequency.

The robustness of our closed-loop system was examined through the method of [7], which analyzed the loop transfer function:

$$L(s) = \frac{K_p s^2 + K_d s + K_i}{s(ms^2 + ds + k)}. \quad \textbf{(25)}$$

The robustness analysis of our closed-loop system using the loop transfer function provided quantitative stability margins, indicating the system's tolerance to gain and phase variations before instability occurs. The calculated gain and phase margins demonstrated that our PID controller design provides sufficient robustness against modeling uncertainties, parameter variations, and external disturbances. The stability margins derived from this transfer function offered a direct measure of the control system's resilience, confirming that the designed controller achieves both performance objectives and stability robustness for the spring-mass-damper system under consideration. The transfer function (equation 15) formed the basis for determining critical frequency-domain stability margins, providing insight into the system's robustness against parameter variations and modeling uncertainties. The stability margins derived from this transfer function offered quantitative measures of how much gain variation and phase shift the system could tolerate while maintaining stability.

2.3 Statistical Framework for Performance Analysis

Building upon the system dynamics simulation framework described before, we developed and implemented a comprehensive statistical analysis approach to compare the performance characteristics of Python and R implementations. All experiments were conducted on a ThinkPad P52s laptop running Fedora Linux 39 (Budgie) x86_64, equipped with an Intel i7-8550U processor (8 threads, 4.000GHz). Given that our numerical simulations primarily involved CPU-bound computations without parallel processing requirements, this standard laptop configuration provided an adequate testing environment representative of typical scientific computing setups in practice.

Our analysis pipeline utilized Python's scientific computing stack, including NumPy for numerical computations [24], SciPy for statistical testing [25], pandas for data management [43], and Seaborn for statistical visualization [28]. Following established practices in performance evaluation [29] and statistical sampling theory [8], we collected $n = 1000$ measurements for each implementation type $i \in$ controlled, open and language $j \in$ [Python, R] after discarding $k = 5$ warm-up runs. These warm-up runs were essential to mitigate the effects of just-in-time compilation and cache warming, as demonstrated by Herho et al. [30, 31, 32].

Our analysis began with fundamental statistical measures for each implementation-language pair (i, j). Following the recommendations of Jain [9] for computer systems performance analysis, we computed the central tendency and dispersion metrics:

$$\mu_{i,j} = \frac{1}{n}\sum_{k=1}^{n} X_{i,j,k}\,, \quad \textbf{(26)}$$

where $\mu_{i,j}$ represents the mean performance measure, n is the number of observations, and $X_{i,j,k}$ is the $k-th$ performance measurement for implementation i and language j.

$$\chi_{i,j} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}\left(X_{i,j,k} - \mu_{i,j}\right)^2}\,, \quad \textbf{(27)}$$

where $\chi_{i,j}$ denotes the standard deviation of the performance measurements.

$$\eta_{i,j} = \frac{\chi_{i,j}}{\mu_{i,j}} \times 100\%\,, \quad \textbf{(28)}$$

where $\eta_{i,j}$ represents the coefficient of variation.

The coefficient of variation in Equation [34] proved particularly valuable for performance comparisons, as it provided a dimensionless measure of variability [10]. Following [11] and

recent performance analysis guidelines [33], we interpreted η below 5% as very consistent performance, 5-10% as good consistency, 10-15% as moderate variability, and above 15% as high variability.

Distribution shape analysis was crucial for understanding performance characteristics and selecting appropriate statistical tests [12]. We computed skewness γ_1 and kurtosis γ_2:

$$\gamma_1 = \frac{1}{n}\sum_{k=1}^{n} \frac{(X_{i,j,k} - \mu_{i,j})^3}{\chi_{i,j}^3}, \tag{29}$$

where γ_1 represents the skewness of the distribution, providing a measure of asymmetry in the performance data. Positive skewness indicates a right-tailed distribution with occasional high outliers, while negative skewness suggests left-tailed distributions with occasional low values

$$\gamma_2 = \frac{1}{n}\sum_{k=1}^{n} \frac{(X_{i,j,k} - \mu_{i,j})^4}{\chi_{i,j}^4} - 3, \tag{30}$$

where γ_2 represents the excess kurtosis of the distribution. These shape parameters were essential for detecting potential performance anomalies and understanding the underlying performance distribution patterns, helping us identify non-standard characteristics in execution time and memory usage distributions across implementations [13].

The normality of distributions was assessed using both Shapiro-Wilk and Kolmogorov-Smirnov tests, as recommended by [34] for their complementary strengths in different sample sizes and distribution types:

$$W = \frac{\left(\sum_{i=1}^{n} \alpha_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{31}$$

where W is the Shapiro-Wilk test statistic, $x_{(i)}$ (with parenthetical index) represents ordered sample values arranged in ascending order, distinct from x_i which denotes the raw sample values or state variables elsewhere in this paper. The constants a_i are derived from the covariance matrix of order statistics, and $\bar{x}$ is the sample mean.

$$D_n = \sup_x |F_n(x) - F(x)|, \tag{32}$$

where D_n is the Kolmogorov-Smirnov test statistic, $F_n(x)$ is the empirical distribution function, and $F(x)$ is the theoretical normal distribution.

For comparing Python and R implementations, we employed non-parametric tests due to their robustness against violations of normality assumptions [14]. The Wilcoxon signed-rank test [35] was used for paired comparisons:

$$W = \sum_{r=1}^{n_r}\left[sgn(x_{2,r} - x_{1,r}).R_r\right], \tag{33}$$

where W is the test statistic, n_r is the number of non-zero differences between paired samples, sgn is the sign function that returns +1, 0, or -1 according to the sign of its argument, and R_r is the rank of the absolute difference. In this context, $x_{2,r}$ and $x_{2,r}$ represent the r-th pair of observations from the two compared implementation methods (Python and R). This paired comparison directly quantifies the significance of performance differences between implementations.

This was complemented by Levene's test [44] for assessing variance homogeneity, chosen for its robustness against non-normality [36]:

$$\Lambda = \frac{(N-k)}{(k-1)} \frac{\sum_{i=1}^{n} N_i (Z_{i.} - Z_{..})^2}{\sum_{i=1}^{k}\sum_{j=1}^{n_i} (Z_{ij} - Z_{i.})^2}, \tag{34}$$

where Λ is the test statistic, N is the total sample size, k is the number of groups, N_i is the sample size of the i-th group, $Z_{ij} = \left|X_{ij} - \widetilde{X}_i\right|$ being the group median, $Z_{i.}$ represents group means of the Z_{ij}, and $Z_{..}$ is the overall mean of Z_{ij}.

Following modern statistical practice [37], we complemented significance testing with effect size analysis using Cohen's d [15]:

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

$$\delta = \frac{\mu_{Python} - \mu_R}{\sqrt{\frac{\chi^2_{Python} + \chi^2_R}{2}}}, \tag{35}$$

where δ is Cohen's d effect size measure, μ_{Python} and μ_R are the means of Python and R implementations respectively, and χ^2_{Python} and χ^2_R are their respective variances.

This measure provided a standardized assessment of practical significance, with effect sizes interpreted according to established guidelines [38] as negligible ($|\delta| < 0.2$), small ($0.2 \leq |\delta| < 0.5$), medium ($0.5 \leq |\delta| < 0.8$), or large ($|\delta| \geq 0.8$).

Performance anomalies were identified using a robust multi-method approach [16], combining three complementary detection methods as recommended by Chandola et al. [39]. The traditional Z-score method:

$$|\zeta| = \left|\frac{x - \mu}{\chi}\right| > 3.5, \tag{36}$$

where $|\zeta|$ is the absolute Z-score, x is the observation value, μ is the mean, and χ is the standard deviation.

This was supplemented by the modified Z-score using median absolute deviation, which offered greater robustness against multiple outliers [40]:

$$|\zeta_{mod}| = \left|\frac{0.6745(x - median)}{MAD}\right| > 3.5, \tag{37}$$

where $|\zeta_{mod}|$ is the modified Z-score, MAD is the median absolute deviation, and 0.6745 is the normalization constant.

The interquartile range method provided a distribution-free approach [17]:

$$x < \mathrm{Q}_1 - 1.5IQR \; or \; x > \mathrm{Q}_3 + 1.5IQR, \tag{38}$$

where Q_1 and Q_3 are the first and third quartiles respectively, and IQR is the interquartile range.

The final set of anomalies $\mathcal{A}$ was determined through consensus voting [18]:

$$\mathcal{A} = \mathcal{A}_\zeta \cup \mathcal{A}_{\mathrm{mod}} \cup \mathcal{A}_{IQR}, \tag{39}$$

where $\mathcal{A}_\zeta$, $\mathcal{A}_{\mathrm{mod}}$, and $\mathcal{A}_{IQR}$ represent the sets of anomalies identified by the Z-score, modified Z-score, and IQR methods respectively.

This statistical framework described in Equations (26)-(39) provided a rigorous foundation for comparing performance characteristics between implementations, adhering to established practices in both statistical methodology and performance analysis [41]. All numerical results were standardized to three decimal places for consistency in technical reporting, as recommended by [19] for reproducible research practices.

Results And Discussion

3.1 System Dynamics

The SMD system exhibited complex dynamic behavior characteristic of underdamped second-order mechanical systems, as evidenced by both mathematical stability analysis and numerical simulations. Figure 3 illustrates the system's temporal evolution under a step force input, while Figure 4 provides insight into the system's state-space behavior through its phase portrait.

The eigenvalue analysis revealed complex conjugate poles at $-0.25 \pm 0.661i$, mathematically confirming the system's underdamped nature. This finding aligns with the calculated damping ratio $\zeta = 0.354$, indicating oscillatory behavior with gradual energy dissipation. The characteristic equation coefficients [1, 0.5, 0.5] satisfied stability criteria across multiple analytical frameworks, including eigenvalue, Routh-Hurwitz, and BIBO stability analyses.

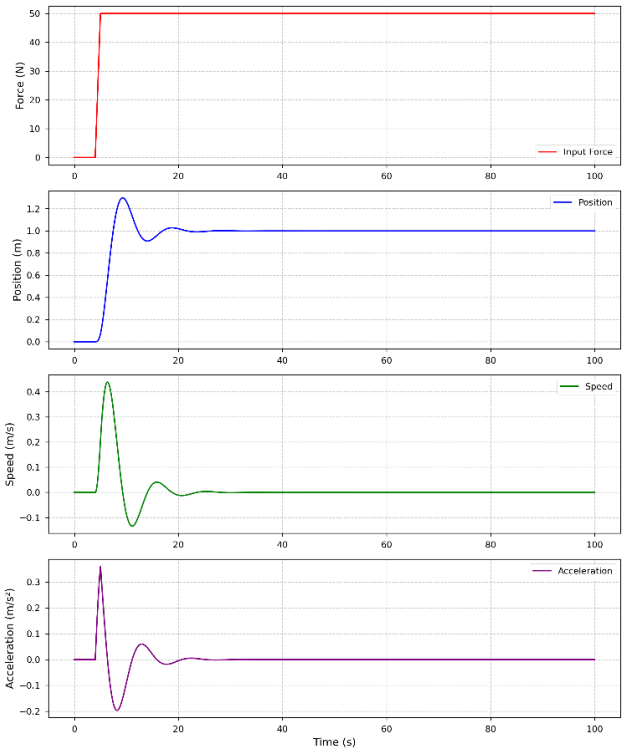

Figure 3. Temporal evolution of the open-loop system response to a 50 N step input. From top to bottom: (a) Applied force showing the transition from 0 to 50 N between t = 4 s and t = 5 s, (b) Position response demonstrating underdamped oscillations converging to 1.000 m, (c) Velocity profile reaching a maximum of 0.4 m/s before dampened oscillations to zero, and (d) Acceleration response highlighting the system's dynamic behavior with peak values of ± 0.2 m/s².

As shown in Figure 3, the system response to the step input force demonstrates characteristic underdamped behavior. The position response (Figure 3b) shows an initial overshoot followed by decaying oscillations before settling to its final value of 1 m. The velocity profile (Figure 3c) exhibits a maximum speed of 0.4 m/s during the initial response phase, with subsequent oscillations diminishing as the system approaches equilibrium. The acceleration response (Figure 3d) shows sharp transitions corresponding to the force application, with peak values of approximately ±0.2 m/s².

The phase portrait in Figure 4 provides additional insight into the system's dynamic behavior. The spiral trajectory illustrates the system's evolution from its initial state at the origin to its final equilibrium point at (1 m, 0 m/s). The decreasing radius of the spiral confirms the system's stability and energy dissipation through damping, while the elliptical shape of the trajectory reflects the continuous exchange between potential and kinetic energy characteristic of spring-mass systems. The simulation results confirm the theoretical stability predictions, with the system achieving its expected final state: position at 1 m, velocity at 0 m/s, and acceleration at 0 m/s².

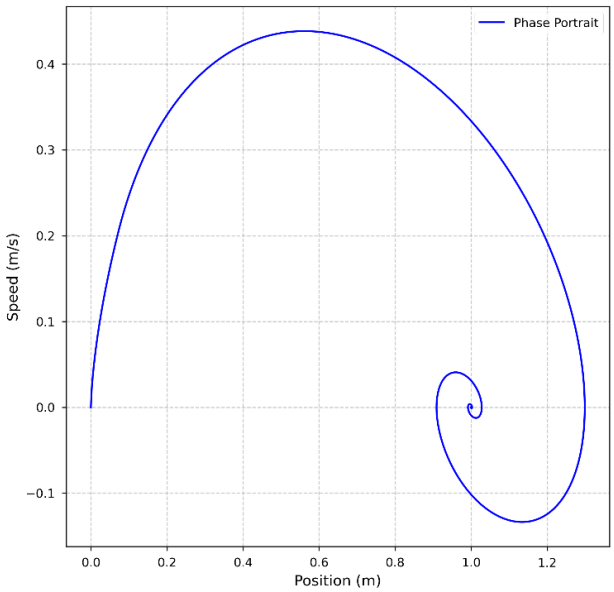

Figure 4. Phase portrait of the open-loop system showing the relationship between position and velocity. The spiral trajectory, beginning from the origin and converging to the equilibrium point (1 m, 0 m/s), illustrates the system's stable focus behavior. The decreasing spiral radius demonstrates the energy dissipation through damping, while the elliptical shape reflects the interchange between potential and kinetic energy.

The implementation of PID control significantly enhanced the system's performance characteristics, introducing more sophisticated dynamic behavior while maintaining stability. Figure 5 presents the comprehensive closed-loop system response, while Figure 6 illustrates the

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

controlled system's state-space behavior through its phase portrait.

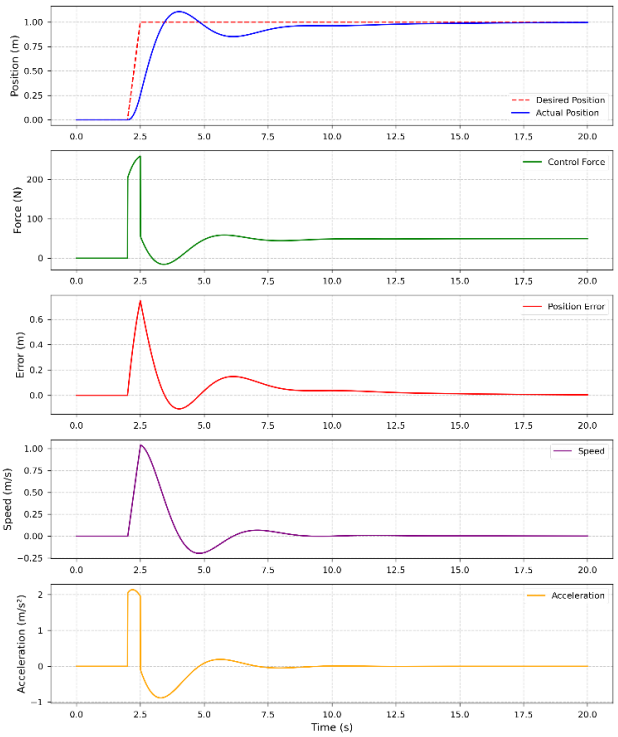

Figure 5. Closed-loop system response with PID control. From top to bottom: (a) Position tracking showing desired (dashed red) and actual (solid blue) trajectories, demonstrating effective reference tracking, (b) Control force profile exhibiting initial peak of 259.2 N followed by smooth regulation, (c) Position error evolution showing maximum deviation of 0.107 m and final error of 0.004 m, (d) Velocity response with peak of 1.04 m/s, and (e) Acceleration profile with maximum of 2.14 m/s².

The closed-loop stability analysis revealed a richer pole constellation compared to the open-loop system, with eigenvalues at -0.5 and $-0.25 \pm 0.661i$. The additional real pole at -0.5, introduced by the controller, enhanced the system's damping characteristics, as evidenced by the increased damping ratio of $\zeta = 0.474$ compared to the open-loop value of 0.354. The characteristic equation coefficients [1, 1.5, 2.5, 0.5] maintained stability across all analytical criteria while enabling improved transient response.

The controlled system demonstrated superior performance metrics, as shown in Figure 5. The position tracking (Figure 5a) achieved a steady-state value of 0.996 m, resulting in a final position error of only 0.004 m (0.4%). The maximum overshoot of 0.107 m (10.7%) occurred during the initial transient response, while the settling time of 12.8 seconds represented a significant improvement over the open-loop behavior. The control force profile (Figure 5b) showed an initial peak of 259.2 N, necessary for rapid response, before settling to a steady-state value that maintained the desired position.

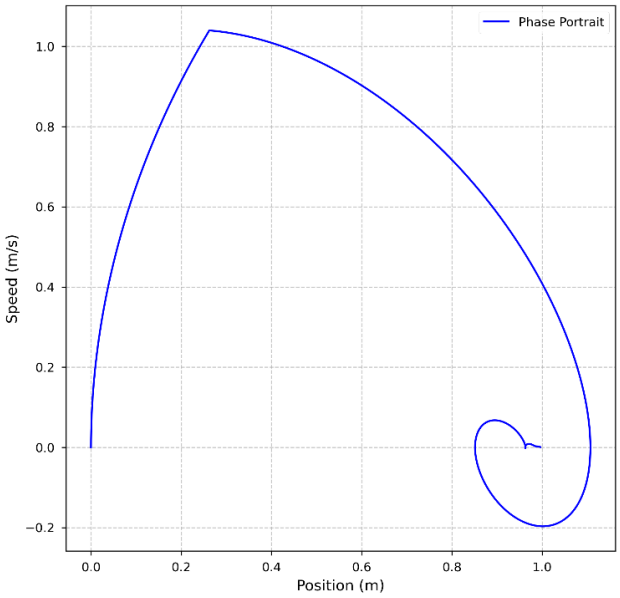

Figure 6. Phase portrait of the closed-loop system illustrating the controlled state-space trajectory. The smooth path from initial conditions to the desired setpoint (1.0 m, 0 m/s) demonstrates the controller's ability to regulate both position and velocity simultaneously. The absence of multiple spirals indicates improved damping compared to the open-loop response.

The phase portrait in Figure 6 reveals the effectiveness of the control strategy in managing the system's states trajectory. Unlike the open-loop case, the controlled system exhibits a more direct path to the desired equilibrium point, with the controller actively shaping the system's dynamic behavior. The single, well-defined trajectory indicates effective energy management by the controller, avoiding the multiple oscillations characteristic of the uncontrolled response.

The RMS position error of 0.124 m provides a quantitative measure of tracking performance throughout the entire operation. The maximum speed of 1.04 m/s and acceleration of 2.14 m/s^2 demonstrate the controller's ability to generate aggressive yet controlled motions when necessary while maintaining system stability. These performance metrics validate the theoretical design of the PID controller with gains K_p = 200 N/m, K_i = 50 N/(ms), and K_d = 100 Ns/m.

3.2 Performance Analysis

Our statistical analysis revealed nuanced and significant differences between Python and R implementations across all metrics and scenarios, evaluated through 1,000 iterations per test. The results show a complex interplay of execution time, memory usage, and stability across controlled and open-loop systems, highlighting the strengths and weaknesses of each platform in scientific computing workflows.

Python demonstrated substantial advantages in execution time performance. In controlled systems, Python achieved a mean execution time of 2.037 seconds (σ = 0.337), significantly outperforming R, which recorded a mean of 5.591 seconds (σ = 0.514). This reduction of 63.566% was statistically significant, confirmed by the Wilcoxon signed-rank test ($p < 0.001$, d = -8.175). However, Python's performance came at the cost of higher variability (CV = 16.553%, categorized as "High variability"), while R showed more consistent execution times (CV = 9.201%, "Good consistency"). Figure 7, left panel, illustrates this contrast with Python exhibiting a broader distribution than R.

In open-loop implementations, Python retained its performance edge with a mean execution time of 3.103 seconds (σ = 0.473) compared to R's 4.223 seconds (σ = 0.667). Although the margin of improvement was reduced to 26.517%, the difference remained statistically significant ($p < 0.001$, d = -1.939). Both platforms exhibited high variability in this scenario (Python CV = 15.227%, R CV = 15.784%). Levene's test further confirmed significant variance differences ($F = 13.157$, $p < 0.001$). The right panel of Figure 7 highlights these trends, showing Python's faster, albeit more variable, performance relative to R.

Memory usage patterns presented a contrasting narrative. In controlled systems, Python consumed slightly more memory, with a mean of 125.461 MB (σ = 0.217) compared to R's 123.583 MB (σ = 0.139), marking a 1.519% increase. This difference was statistically significant ($p < 0.001$) with a large effect size (d = 10.305). However, both platforms demonstrated exceptional stability, as reflected by CV values of 0.173% for Python and 0.112% for R. These results suggest that Python's higher baseline interpreter memory footprint accounts for the slight difference.

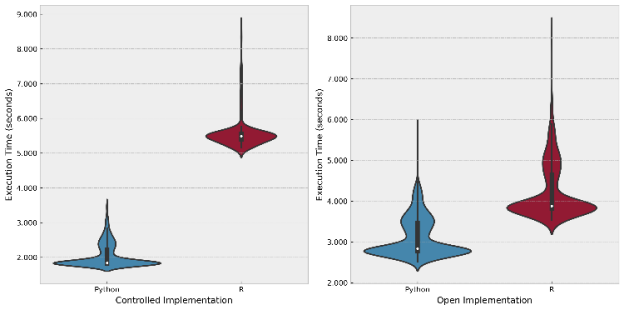

Figure 7. Violin plots of execution time distributions for Python and R under controlled and open-loop implementations. The left panel represents controlled implementations, while the right panel shows open-loop implementations.

In open-loop systems, memory usage between the platforms converged. Python recorded a mean of 132.965 MB (σ = 0.224), while R reached 133.020 MB (σ = 0.112). The negligible difference of -0.042% was statistically significant ($p < 0.001$), though with a small effect size (d = -0.312), indicating practical equivalence. Both implementations maintained highly consistent memory utilization, with CV values below 0.2%. Figure 8, right panel, illustrates this near-parity in memory usage distributions.

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

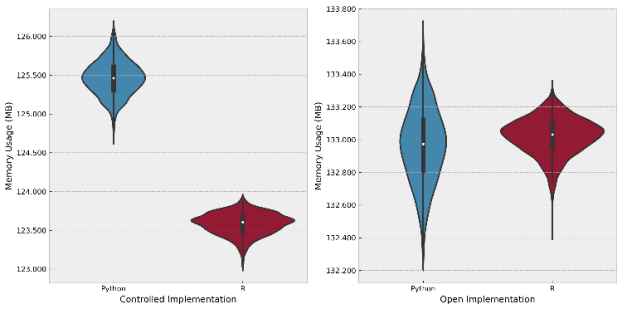

Figure 8. Violin plots of memory usage distributions for Python and R under controlled and open-loop implementations. The left panel represents controlled implementations, while the right panel shows open-loop implementations.

Distribution shape analysis provided additional insights. Execution time data showed significant right skewness in both platforms, with Python exhibiting skewness ≥ 1.094 and R exceeding 3.369 in controlled implementations. Non-normality tests (Shapiro-Wilk $p < 0.001$) confirmed deviations from Gaussian behavior, as shown by broader tails in Figure 7. Memory usage data, on the other hand, displayed symmetric or moderately skewed distributions depending on the scenario, reflecting greater consistency in resource allocation.

Anomaly counts varied significantly across configurations. For controlled execution times, Python recorded a high anomaly rate (33.0% of data) compared to R's 10.6%, reflecting the variability induced by Python's dynamic optimization strategies. Conversely, in open-loop memory usage, R exhibited more anomalies (1.8%) compared to Python's 0.4%, underscoring R's sensitivity to subtle system conditions in memory-intensive scenarios.

Overall, Python's execution speed makes it ideal for computationally intensive tasks, especially in controlled systems where performance is critical. However, its variability and higher anomaly rates suggest careful consideration for applications requiring extreme precision and stability. R's strength lies in its consistent memory usage and lower variability, making it a better choice for scenarios where stability and resource predictability are paramount. These results underscore the importance of aligning platform selection with specific computational requirements to optimize performance and efficiency.

For example, Masini and Bientinesi [43] emphasizes Python's broader ecosystem and optimization strategies as critical for scalable applications, while Raschka et al. [44] highlight R's robustness for statistical workflows. These findings align with earlier research by Watson et al. [45], suggesting platform-specific trade-offs in computational reliability.

Conclusions

This study provides a comprehensive evaluation of Python and R for simulating and controlling SMD systems, with a focus on computational performance and numerical accuracy. Python exhibited a clear advantage in execution speed, achieving reductions of up to 63.57% in mean runtime compared to R in controlled system simulations. However, this efficiency came with trade-offs, including higher variability in execution times and a greater anomaly rate, reflecting the platform's dynamic optimization strategies. Conversely, R demonstrated superior consistency in execution and memory usage, making it more suitable for scenarios prioritizing stability and resource predictability. These findings underscore the distinct strengths of each platform and the importance of selecting the right tool for specific computational needs.

It is important to note that while Python and R demonstrated significant differences in computational performance, these differences did not affect the physical behavior or stability characteristics of the simulated system. Both platforms produced identical system dynamics when using the same numerical solver (LSODA) with equivalent tolerance settings, confirming that the 63.57% execution speed advantage of Python represents purely computational efficiency rather than differences in simulation accuracy or dynamic response. The time required for the controlled

system to reach stability (approximately 12.8 seconds) remained consistent across both implementations, validating the platform-independent nature of the underlying mathematical model. This consistency in physical results despite performance differences further strengthens our confidence in the robustness of both implementations for scientific computing applications.

The implementation of PID control significantly enhanced system dynamics, yielding improved position tracking and stability. With a final position error of just 0.4% and enhanced damping characteristics, the controlled system achieved superior performance metrics over the open-loop configuration. The controlled system's steady-state precision and reduced settling time validate the PID controller's design and its effectiveness in managing transient responses. Overall, this work bridges theoretical stability analysis with empirical performance insights, providing a valuable resource for researchers in computational dynamics and control systems. By leveraging the capabilities of open-source platforms, this research promotes reproducibility and transparency in the numerical study of dynamic systems.

Acknowledgements

Code and Data Avalability. All code used for numerical simulations, statistical analyses, and the complete dataset generated during this study are openly available in the GitHub repository at https://github.com/sandyherho/smdCompare.

Funding. This work was supported by the Dean's Distinguished Fellowship, University of California, Riverside, in 2023.

Declarations

Conflict of interest. The authors declare there is no conflict.

Competing interests. Authors do not have any competing financial interest to declare.

References

[1] Ogata, K.: Modern Control Engineering. Prentice Hall, Upper Saddle River, NJ, USA (2010).
[2] Khalil, H.K.: Nonlinear Systems. Prentice Hall, Upper Saddle River, New Jersey, USA (2002).
[3] Dorf, R.C., Bishop, R.H.: Modern Control Systems. Prentice Hall, Upper Saddle River, NJ, USA (2011).
[4] Nise, N.S.: Control Systems Engineering. John Wiley & Sons, New York City, NY, USA (2020).
[5] Franklin, G.F., Powell, J.D., Emami-Naeini, A.: Feedback Control of Dynamic Systems. Pearson, Upper Saddle River, NJ, USA (2015).
[6] Åström, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press, Princeton, NJ, USA (2008).
[7] Chen, C.-T.: Linear System Theory and Design. Oxford University Press, Oxford, UK (1995).
[8] Thompson, S.K.: Sampling. Wiley Series in Probability and Statistics, New York City, NY, USA (2012).
[9] Jain, R.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, New York, USA (1991).
[10] Brown, R.J.C., Brown, R.F.C.: Statistical Analysis of Measurement Data. Royal Society of Chemistry, London, United Kingdom (1998).
[11] Hoaglin, D.C., Mosteller, F., Tukey, J.W.: Understanding Robust and Exploratory Data Analysis. Wiley-Interscience, Hoboken, NJ, USA (2000).
[12] Montgomery, D.C.: Design and Analysis of Experiments. John Wiley & Sons, New York City, NY, USA (2017).
[13] Bulmer, M.G.: Principles of Statistics. Dover Publications, New York, USA (1979).
[14] Siegel, S.: Nonparametric Statistics for the Behavioral Sciences. McGraw-Hill, New York City, NY, USA (1956).
[15] Cohen, J.: Statistical Power Analysis for the Behavioral Sciences. Routledge, New York City, NY, USA (1988). https://doi.org/10.4324/9780203771587.
[16] Iglewicz, B., Hoaglin, D.C.: Volume 16: How to Detect and Handle Outliers. ASQC Quality Press, Milwaukee, WI, USA (1993).
[17] Tukey, J.W.: Exploratory Data Analysis. Addison-Wesley, Reading, MA, USA (1977).
[18] Aggarwal, C.C.: Outlier Analysis. Springer, New York City, NY, USA (2013). https://doi.org/10.1007/978-1-4614-6396-2.
[19] Altman, D.G., Machin, D., Bryant, T.N., Gardner, M.J.: Statistics with Confidence: Confidence Intervals and Statistical Guidelines. BMJ Books, London, UK (2013).

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia

Original Article

[20] Åström, K.J., Hägglund, T.: Advanced PID Control. ISA-The Instrumentation, Systems, and Automation Society (2006).

[21] Irawan, D.E., Pourret, O., Besançon, L., Herho, S.H.S., Ridlo, I.A., Abraham, J.: Post-Publication Review: The Role of Science News Outlets and Social Media. Annals of Library and Information Studies 71, 465–474 (2024). https://doi.org/10.56042/alis.v71i4.1425.

[22] Fraser, N., Brierley, L., Dey, G., Polka, J.K., Pálfy, M., Nanni, F., Coates, J.A.: The evolving role of preprints in the dissemination of COVID-19 research and their impact on the science communication landscape. PLoS Biology 19(4), 3000959 (2021). https://doi.org/10.1371/journal.pbio.3000959.

[23] Sugimoto, C.R., Work, S., Lariviere, V., Haustein, S.: Scholarly use of social media and altmetrics: A review of the literature. Journal of the Association for Information Science and Technology 68(9), 2037–2062 (2017). https://doi.org/10.1002/asi.23833.

[24] Harris, C., Millman, K., Walt, S., Gommers, R., Virtanen, P., Cournapeau, D., et al.: Array Programming with NumPy. Nature 585(7825), 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2.

[25] Virtanen, P., Gommers, R., Oliphant, T.E., et al.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17(3), 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2.

[26] Soetaert, K., Petzoldt, T., Setzer, R.W.: Solving Differential Equations in R: Package deSolve. Journal of Statistical Software 33(9), 1–25 (2010). https://doi.org/10.18637/jss.v033.i09.

[27] Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L.D., et al.: Welcome to the Tidyverse. Journal of Open Source Software 4(43), 1686 (2019). https://doi.org/10.21105/joss.01686.

[28] Waskom, M.L.: Seaborn: Statistical Data Visualization. Journal of Open-Source Software 6(60), 3021 (2021). https://doi.org/10.21105/joss.03021.

[29] Georges, A., Buytaert, D., Eeckhout, L.: Statistically Rigorous Java Performance Evaluation. ACM SIGPLAN Notices 42(10), 57–76 (2007). https://doi.org/10.1145/1297105.1297033.

[30] Herho, S., Anwar, I., Herho, K., Dharma, C., Irawan, D.: COMPARING SCIENTIFIC COMPUTING ENVIRONMENTS FOR SIMULATING 2D NON-BUOYANT FLUID PARCEL TRAJECTORY UNDER INERTIAL OSCILLATION: A PRELIMINARY EDUCATIONAL STUDY. Indonesian Physical Review 7(3), 451–468 (2024). https://doi.org/10.29303/ipr.v7i3.335.

[31] Herho, S., Fajary, F., Herho, K., Anwar, I., Suwarman, R., Irawan, D.: Reappraising Double Pendulum Dynamics across Multiple Computational Platforms. CLEI Electronic Journal 28(1) (2025). https://doi.org/10.19153/cleiej.28.1.10.

[32] Herho, S., Kaban, S.N., Irawan, D.E., Kapid, R.: Efficient 1D Heat Equation Solver: Leveraging Numba in Python. Eksakta: Berkala Ilmiah Bidang MIPA 25(2), 126–137 (2024). https://doi.org/10.24036/eksakta/vol25-iss02/487.

[33] Mostafavi, S., Hakami, V., Paydar, F.: Performance Evaluation of Software Defined Networking Controllers: A Comparative Study. Computer and Knowledge Engineering 2(2), 63–73 (2020). https://doi.org/10.22067/cke.v2i2.84917.

[34] Razali, N.M., Wah, Y.B.: Power Comparisons of Shapiro-Wilk, Kolmogorov Smirnov, Lilliefors and Anderson-Darling Tests. Journal of Statistical Modeling and Analytics 2(1), 21–33 (2011).

[35] Wilcoxon, F.: Individual Comparisons by Ranking Methods. Biometrics Bulletin 1(6), 80–83 (1945). https://doi.org/10.2307/3001968.

[36] Brown, M.B., Forsythe, A.B.: Robust Tests for the Equality of Variances. Journal of the American Statistical Association 69(346), 364–367 (1974). https://doi.org/10.1080/01621459.1974.10482955.

[37] Sullivan, G.M., Feinn, R.: Using Effect Size or Why the P Value is Not Enough. Journal of Graduate Medical Education 4(3), 279–282 (2012). https://doi.org/10.4300/JGME-D-12-00156.1.

[38] Sawilowsky, S.S.: New Effect Size Rules of Thumb. Journal of Modern Applied Statistical Methods 8(2), 597–599 (2009). https://doi.org/10.22237/jmasm/1257035100.

[39] Chandola, V., Banerjee, A., Kumar, V.: Anomaly Detection: A Survey. ACM Computing Surveys 41(3), 1–58 (2009). https://doi.org/10.1145/1541880.1541882.

[40] Rousseeuw, P.J., Croux, C.: Alternatives to the Median Absolute Deviation. Journal of the American Statistical Association 88(424), 1273–1283 (1993). https://doi.org/10.1080/01621459.1993.10476408.

[41] Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F.: Producing Wrong Data Without Doing Anything Obviously Wrong! ACM SIGARCH Computer Architecture News 37(1), 265–276 (2009). https://doi.org/10.1145/2528521.1508275.

[42] Tippmann, S.: Programming Tools: Adventures with R. Nature 517(7532), 109-110 (2015). https://doi.org/10.1038/517109a.

[43] McKinney, W.: Data Structures for Statistical Computing in Python. In: Walt, S., Millman, J. (eds.) Proceedings of the 9th Python in Science Conference, pp. 51–56 (2010). https://doi.org/10.25080/majora-92bf1922-00a.

[44] Levene, H.: Robust Tests for Equality of Variances. In: Olkin, I. (ed.) Contributions to Probability and Statistics, pp. 278–292. Stanford University Press, Stanford, California, USA (1960).

[45] Masini, S., Bientinesi, P.: High-Performance Parallel Computations Using Python as High-Level Language. In: Guarracino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M. (eds.) Euro-Par 2010 Parallel Processing Workshops, pp. 541–548. Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21878-1_66

[46] Raschka, S., Patterson, J., Nolet, C.: Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence.

Information 11(4), 193 (2020). https://doi.org/10.3390/info11040193

[47] Watson, A., Babu, D.S.V., Ray, S.: Sanzu: A data science benchmark. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 263–272 (2017). https://doi.org/10.1109/BigData.2017.8257934

[48] Momcheva, I., Tollerud, E.: Software Use in Astronomy: An Informal Survey. arXiv preprint arXiv:1507.03989 (2015).

Published by: Lembaga Penelitian dan Pengabdian Masyarakat (LPPM)
Institut Teknologi Sumatera, Lampung Selatan, Indonesia